Modeling, Identification and Control, Vol. 31, No. 3, 2010, pp. 107–121, ISSN 1890–1328

Model-based optimizing control and estimation using Modelica models

L. Imsland [1] P. Kittilsen [2] T. S. Schei [2]

[1] *Department of Engineering Cybernetics, Norwegian University of Science and Technology, N-7491 Trondheim, Norway. E-mail:* lars.imsland@itk.ntnu.no

[2] *Cybernetica AS, N-7032 Trondheim, Norway. E-mail:* {pal.kittilsen,tor.steinar.schei}@cybernetica.no

Abstract

This paper reports on experiences from case studies in using Modelica/Dymola models interfaced to control and optimization software, as process models in real time process control applications. Possible applications of the integrated models are in state- and parameter estimation and nonlinear model predictive control. It was found that this approach is clearly possible, providing many advantages over modeling in low-level programming languages. However, some effort is required in making the Modelica models accessible to NMPC software.

Particular consideration is given to implementation of gradient computation for real-time dynamic optimization, where the dynamic models can be Modelica models. Analytical methods for gradient computation based on sensitivity integration are compared to finite difference-based methods. A case study reveals that analytical methods outperform finite difference-methods as the number of inputs and/or input blocks increases.

Keywords: Non-linear model predictive control, state estimation, Modelica, offshore oil- and gas production, gradient computation

1 Introduction

Nonlinear model predictive control (NMPC) is an advanced control technology that enables the use of mechanistic multi-disciplinary process models in achieving process control objectives (economical, safety, environmental). NMPC algorithms formulate an 'open-loop' constrained dynamic optimization problem, which is re-solved and re-implemented at regular intervals to combine the advantage of the optimal control solution with the feedback achieved through updated information (measurements and estimated states and parameters).

Although *linear* MPC (based on linear, typically empirical, process models) is prevalent, it is seen that in many cases, MPC based on nonlinear process models (NMPC), with models derived from first principles and process knowledge, is advantageous or even necessary to achieve better control performance over varying operating conditions (due, for example, to varying product specifications or large process disturbances). In addition to the use of nonlinear process models, another important aspect with NMPC based on models from first principles, is that nonlinear state estimation is an essential part of the control system.

NMPC has received considerable attention in academia, especially in terms of optimization methods (Biegler, 2000) and requirements for stability of the resulting closed loop (Findeisen et al., 2003). However, when it comes to industrial application, use of NMPC clearly has an unfulfilled potential, although an increasing number of applications are being reported, especially in polymerization processes (Foss and Schei, 2007; Qin and Badgwell, 2003).

doi:10.4173/mic.2010.3.3

One important reason for the limited practical use of NMPC, is the substantial time and effort required for developing, validating and maintaining nonlinear process models that are valid over a wide operating range. Importantly, but sometimes overlooked, these models should at the same time be suitable for optimization, in terms of issues such as complexity and smoothness. An important step towards less costly model development is the use of advanced modeling environments, which promote model structure, model reuse and model maintenance through equation-oriented modeling languages, object orientation and hierarchical composition of sub-models.

Literature reveals some effort towards using advanced process modeling environments in a practical dynamical optimization setting, e.g. Lang and Biegler (2007), where the gPROMS system from Process Systems Enterprise Limited is connected to a software environment for dynamic optimization. However, the impression remains that this is very much a developing area.

The use of such models is not limited to NMPC in real-time process control settings. One can envision many types of real-time model-based applications using such models, ranging from data reconciliation, estimation (states, parameters, disturbances, soft-sensing) for monitoring and control, to advisory operator support systems and finally to NMPC. One can argue that a complete NMPC installation involves the other applications mentioned, such that if Modelica models can be used for NMPC, the other applications follow naturally.

The aim of this paper is to discuss requirements, challenges, opportunities, and experiences from using an advanced modeling environment, in particular Dymola/Modelica, for developing models that are used in model-based process control applications.

The paper is structured as follows: We start by discussing some basic elements of an NMPC system, and then go into some detail on gradient computation for a class of NMPC optimization algorithms often referred to as sequential (or single-shooting) NMPC optimization. We discuss some implementation aspects in Section 4 and 5, before we go on by presenting some studies on state- and parameter estimation, and NMPC from the oil and gas production industry. Performance of gradient computation algorithms are illustrated in Section 8. We end the paper by discussing some of our experiences.

This paper is a combined and revised version of Imsland et al. (2008) and Imsland et al. (2009).

2 Elements in NMPC

2.1 Models

We assume that a model of the physical plant we want to control is implemented in Modelica. The underlying mathematical representation could be either as (hybrid) ordinary differential equations (ODEs) or differential-algebraic equations (DAEs), but here we assume, mainly for simplicity, that it is formulated as a (piecewise) continuous ODE:

$$\dot{x} = f(x,u,p), \quad y = h(x,u,p), \quad z = g(x,u,p), \tag{1}$$

where x are states, u are manipulated inputs, p are parameters (which might be candidates for online estimation), y are measured outputs and z are controlled (not necessarily measured) outputs.

Note that typically, we will in addition to manipulated inputs also have other (measured) inputs that in essence make the system time-variant. However, we will employ discrete-time state estimation and NMPC formulations and are therefore only interested in integration over one sample interval where these inputs typically are assumed constant.

For the same reason, we are only interested in the solution of (1) in the sense that it is used to calculate states and outputs at the next sampling instant. That is, we are interested in the discrete-time system

$$x_{k+1} = x_k + \int_{t_k}^{t_{k+1}} f(x(\tau), u_k, p_k)\mathrm{d}\tau, \tag{2a}$$

$$y_k = h(x_k, u_{k-1}, p_{k-1}), \tag{2b}$$

$$z_k = g(x_k, u_{k-1}, p_{k-1}). \tag{2c}$$

The integration involved is in general solved by ODE solver routines.

For NMPC, we will use the above system for prediction. In prediction for NMPC, we are not concerned with the measured outputs, and therefore, with abuse of notation, we will write the time-varying discrete-time NMPC predictor system as

$$x_{k+1} = f_k(x_k, u_k), \quad z_k = g_k(x_k, u_{k-1}). \tag{3}$$

Similarly, for state estimation we use in principle the same discrete-time dynamic model with y_k as output.

2.2 Model issues

Using equation-based modeling environments such as those based on Modelica, one generally ends up with differential-algebraic equation systems (DAEs). In Dymola, there are implemented algorithms for reformulation (symbolic transformation) of the DAE system such that it from the outside looks like an ODE system, but

where the evaluation of the right hand side in general requires the solution of some nonlinear equation systems. The reformulation ensures that these nonlinear equation systems are as small, and hence as efficiently solved, as possible. However, the solution of them is based on iterative, local methods, such that it can in general take many iterations to find an acceptable solution, and worse, one is not always guaranteed to find a solution at all. (Although for well-behaved models, one normally finds a solution in few iterations.)

Another issue is that the right hand side might be discontinuous in its arguments. If this is the case, the solvers used to solve the (apparent) ODE above, must be able to handle discontinuities. Moreover, the system will often be stiff, calling for implicit methods with variable step lengths.

Apart from any possible discontinuities, the above issues (DAEs, stiffness, variable step lengths) do not in principle imply any problems using Modelica/Dymola models with NMPC tools. Nevertheless, efficiency and robustness issues may change the picture. Simulation in an NMPC system involves frequent resetting of system parameters (initial states, inputs and estimated parameters), which for DAEs in general requires online re-solving of the nonlinear equation set. For the ODEs exported by Dymola, it leads to frequent re-solving of the 'hidden' nonlinear equation sets.

If we can ensure that the model is a 'real' ODE (without nonlinear equation sets), this is avoided, resulting in increased speed and robustness. There is no direct help in Dymola to avoid the nonlinear equation sets leading to a DAE system, but the reporting when translating models helps to identify where these nonlinear equations are.

Additionally, ensuring that the model is continuous, means that we can use more efficient solvers that do not have to handle discontinuities.

These issues require more effort during the modeling, and also imply that one often cannot apply other (library, customer) models directly. Nevertheless, the issues are important: In our experience, it is a key aspect of a successful implementation of an NMPC system to find the correct balance between computational complexity of the model/simulation and required model accuracy. Required model accuracy is not easily defined in general, but relates to the specific control objectives of the particular process. In this respect, more complex models are not necessarily more accurate.

When building models from physics, one typically ends up with stiff equation systems, which require implicit solvers with variable step sizes to be solved efficiently. Analytical model Jacobians can be be used in implicit solvers to speed up computation significantly.

2.3 NMPC optimization

The NMPC optimization problem is a dynamic optimization problem, usually discretized to have a finite number of optimization variables (manipulated variables), that must be solved at regular (sampling) instants. The first part of the optimal solution – usually the first sample interval – is implemented to the process, before the dynamic optimization problem is resolved before the next sample instant. The optimization problem is using updated process information from a state estimation algorithm.

The optimization problem to be solved at time t, with available state estimate $\hat{x}(t)$, may look something like this, after a piecewise constant parameterization of future manipulated variables (u) over an horizon N:

$$\min_{u_0,u_1,\ldots,u_{N-1}} \sum_{k=0}^{N-1} F(x_{k+1},u_k)$$

$$\text{subject to } \begin{cases} x_{k+1}-f(x_k,u_k) & = 0,\ k=0,\ldots,N-1, \\ x_0 & = \hat{x}(t), \\ h_x(x_k) & \geq 0,\ k=1,\ldots,N, \\ h_u(u_k) & \geq 0,\ k=0,\ldots,N-1. \end{cases}$$

The functions h_x and h_u represent constraints on states (or controlled variables) and manipulated variables.

In most cases, the (discretized) dynamic optimization problem is solved using numerical algorithms based on sequential quadratic programming (SQP). A SQP method is an iterative method which at each iteration makes a quadratic approximation to the objective function and a linear approximation to the constraints, and solves a QP to find the search direction. Then a linesearch is performed along this search direction to find an acceptable next iterate, and the process is repeated until convergence (or time has run out). General purpose SQP solvers may be applied to NMPC optimization, but it is in general advantageous to use tailor-made SQP algorithms for NMPC applications.

The main approaches found in the literature are usually categorized by how the dynamic optimization problem is discretized/parametrized. The most common method is perhaps the *sequential* approach (Li and Biegler, 1989), which at each iteration simulates the model using the current value of the optimization variables $(u_0, u_1, \ldots, u_{N-1})$ to obtain the gradient of the objective function (and possibly the Hessian), thus effectively removing the model equality constraints and the states $x_1, x_2, \ldots, x_N$ as optimization variables. In this paper, a sequential approach is assumed.

Other methods are the *simultaneous* approach (Biegler et al., 2002), and the *multiple shooting* approach (Bock et al., 2000). All methods

will benefit from having analytical Jacobians available from a modeling system.

2.4 State estimation

Model-based control typically apply some estimation scheme, for instance a sigma-point extended Kalman-filter approach or moving horizon estimation (Schei, 2008). The tasks of the estimation in a model-based control context, are to

- consolidate measurements to obtain a best estimate of the process state,
- estimate unknown and changing parameters (adaptation), and
- achieve zero steady state error in the desired controlled variables (integral control).

Nonlinear state-, disturbance- and parameter estimation are essential for NMPC implementations, but are also important in other settings than purely control-related, such as monitoring and surveillance, and static optimization/RTOs.

Estimation based on Kalman filter algorithms has become tremendously widespread over the last almost 50 years. Other types of estimation algorithms also exist, but are much less used. Kalman filter algorithms for nonlinear systems (Extended Kalman Filters, EKFs) have traditionally been based on analytical model linearizations, but over the last years, it is seen that using divided differences (or similarly, Unscented Kalman Filtering (UKF) approaches) in many cases provides better performance than linearization-based EKF.

Importantly, the perturbation schemes used in connection with covariance update by divided difference-approaches (including UKFs) obtain information beyond linearization. Thus, for these cases, availability of analytical Jacobians from the model is not necessarily an advantage (unless it speeds up simulation). On the other hand, for estimation schemes based on linearizations (e.g. traditional EKF), or estimation based on numerical optimization (e.g. Moving Horizon Estimation (MHE)-approaches, taking inequality constraints into consideration), analytical Jacobians can be exploited.

The work in this paper uses an EKF implementation based on divided differences (both DD1 (Schei, 1997) and DD2 (Nørgaard et al., 2000), in the notation of Nørgaard et al. (2000)). For further information and discussion, see also Schei (2007).

3 Gradient computation in sequential NMPC optimization

As gradient computations are by far the most time-consuming part of a sequential NMPC optimization algorithm, this section gives a discussion on algorithms for this with an emphasis on issues related to choice of an underlying modeling system like Modelica. Some implementation results corresponding to this section are given in Section 8.

3.1 Simplified NMPC optimization problem

For the purpose of illustration, we formulate in this section a simplified discrete-time NMPC optimization problem using the model (3). We assume the desired operating point $(x, u) = (0, 0)$ is an equilibrium ($f_k(0, 0) = 0$, $g_k(0, 0) = 0$), and we minimize at each sample (using present measured/estimated state x_0 as initial state for predictions) the objective function

$$J(x_0, u_0, u_1, \ldots, u_{N-1}) = \frac{1}{2} \sum_{i=0}^{N-1} z_{i+1}^{\mathsf{T}} Q z_{i+1} + u_i^{\mathsf{T}} R u_i$$

over future manipulated inputs u_i, where z_i are computed (predicted) from (3), and Q and R are weighting matrices. Importantly, the future behavior is optimized subject to constraints:

$$\begin{aligned} z_{\min} \leq z_k = g_k(x_k, u_{k-1}) \leq z_{\max}, \quad k = 1, \ldots, N \\ u_{\min} \leq u_k \leq u_{\max}, \quad k = 0, \ldots N-1. \end{aligned}$$

The first input u_0 is then applied to the plant.

It is important to note that the problem formulation used in this section is simplistic. For the sake of brevity and with little loss of generality, it does not contain features usually contained in NMPC software packages, such as:

- Features related to non-regulation problems (for instance control of batch processes).
- Input blocking (for efficiency).
- Incidence points (for efficiency and feasibility).
- Control horizon longer than input horizon.
- End-point terminal weight/region (in regulation, for efficiency/stability).
- Input moves instead of inputs as optimization variables.

Further details about such issues can be found in MPC textbooks, for instance Rawlings and Mayne (2009); Maciejowski (2001).

3.2 The sensitivity (step/impulse response) matrix

As explained above, sequential SQP approaches to NMPC optimization sequentially simulates and optimizes. The simulation part should calculate the objective function- and constraint gradients with respect to the optimization variables u_i. This is typically done via the *step response matrix*, or as in the simplified exposition here, the impulse response matrix. To avoid confusion, we will mostly refer to this in the following as the (NMPC) sensitivity matrix.

Rewrite the objective function by stacking future inputs and outputs as

$$J(x_0, \mathbf{u}) = \frac{1}{2}\left(\mathbf{z}^\mathsf{T}\mathbf{Q}\mathbf{z} + \mathbf{u}^\mathsf{T}\mathbf{R}\mathbf{u}\right),$$

where $\mathbf{u} = (u_0, u_1, \ldots, u_{N-1})$, $\mathbf{z} = (z_1, \ldots, z_N)$, $\mathbf{Q} = \text{blkdiag}\{Q\}$ and $\mathbf{R} = \text{blkdiag}\{R\}$. The gradient of the objective function is

$$\frac{\partial J}{\partial \mathbf{u}} = \mathbf{z}^\mathsf{T}\mathbf{Q}\frac{\partial \mathbf{z}}{\partial \mathbf{u}} + \mathbf{u}^\mathsf{T}\mathbf{R}.$$

Similarly, the linearization of the output constraints are also given by the matrix $\Phi = \frac{\partial \mathbf{z}}{\partial \mathbf{u}}$, which we call the sensitivity matrix (which in this case is the truncated impulse response matrix).

That is, once we have calculated the sensitivity matrix Φ, we can easily evaluate the objective function- and constraints gradients in sequential NMPC optimization. From this, one can argue that gradient computation in sequential NMPC is mostly about efficient computation of the sensitivity matrix.

The rest of this section treats calculation of the sensitivity matrix by finite differences, and by forward ODE sensitivity integration. We remark that one could also use adjoint sensitivity methods for calculating the desired NMPC gradients (Jørgensen, 2007; Ringset et al., 2010). However, for NMPC problems with a significant number of constraints, this is likely to be less efficient than forward methods.

3.3 NMPC sensitivity matrix by finite differences

Finding the sensitivity matrix by finite differences is achieved by in turn perturbing each element of all input vectors u_i and simulate to find the response in the z_js. The perturbation is typically either one-sided (forward finite differences) or two-sided (central finite differences), the latter taking about twice the time but being somewhat more accurate (Nocedal and Wright, 2006).

3.4 NMPC sensitivity matrix by sensitivity integration

Assuming we have a time-varying linearization of (3) along the trajectories:

$$x_{k+1} = A_k x_k + B_k u_k, \tag{4a}$$
$$z_k = C_k x_k + D_k u_{k-1}, \tag{4b}$$

we can calculate the sensitivity matrix which in this simple case is as shown in eq. (5) on the bottom of the page. (The sensitivity matrix shown there is the impulse response matrix, the step response matrix is the cumulative sum of the columns of the impulse response matrix, from right to left.) To find the linearization (4), it is usually most practical to calculate sensitivities of the solution of (1) with respect to initial values $x(0) = x_k$ and inputs u_k (assumed constant over each sample interval). Stacking these sensitivities in matrices S and W, they are given by the following matrix ODEs (Hairer et al., 1993):

$$S := \frac{\partial x}{\partial u_k}: \quad \dot{S} = \frac{\partial f}{\partial x}S + \frac{\partial f}{\partial u}, \quad S(0) = 0, \tag{6a}$$
$$W := \frac{\partial x}{\partial x_k}: \quad \dot{W} = \frac{\partial f}{\partial x}W, \quad W(0) = I. \tag{6b}$$

We will only be interested in the sensitivities at the end of each sample interval, which are the system matrices in (4a):

$$B_k := \frac{\partial x_{k+1}}{\partial u_k} = S_{k+1}, \quad A_k := \frac{\partial x_{k+1}}{\partial x_k} = W_{k+1}. \tag{7}$$

Defining also $C_k := \frac{\partial z_k}{\partial x_k}$, $D_k := \frac{\partial z_k}{\partial u_{k-1}}$, we get the linearized (LTV) system above. The required Jacobian matrices

$$\frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial u}, \quad \frac{\partial g}{\partial x}, \quad \frac{\partial g}{\partial u}$$

$$\begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_N \end{pmatrix} = \begin{pmatrix} C_1B_0 + D_1 & 0 & 0 & 0 & \cdots \\ C_2A_1B_0 & C_2B_1 + D_2 & 0 & 0 & \cdots \\ C_3A_2A_1B_0 & C_3A_2B_1 & C_3B_2 + D_3 & 0 & \cdots \\ C_4A_3A_2A_1B_0 & C_4A_3A_2B_1 & C_4A_3B_2 & C_4B_3 + D_4 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{pmatrix} \tag{5}$$

can be found from finite differences, symbolically (for instance from a Dymola model), or by automatic differentiation methods. The two latter should be preferred.

The sensitivity ODEs (6) are solved together with (2a) by ODE solvers. Although the size of S and W might be large, the fact that the systems (6) have a block-diagonal Jacobian with the individual blocks being the Jacobian of (2a) can and should be exploited in ODE solvers, leading to efficient computation of the ODE sensitivities (Schlegel et al., 2004; Hindmarsh and Serban, 2006). (For systems with a very large number of states and relatively few inputs/outputs, it might be more efficient to directly integrate the elements in the sensitivity matrix, and thus avoiding calculation of state sensitivities.)

It is important to note that the above is described for zero order hold and no input blocking. For efficient implementation, it is essential to exploit input blocking in the sensitivity computations. For further discussion about gradient computations, we refer also to Ringset et al. (2010).

4 Implementation

A software package for model-based estimation and control (NMPC) will typically include an *offline* part for model fitting (parameter optimization) to data, data-based testing of estimation and simulation-based testing of NMPC (including estimation), and an *online* part for a complete NMPC real-time solution (including estimation). The workflow in taking a parametrized model (implemented in Modelica/Dymola, or 'by hand' in lower level languages such as C) to online application is attempted illustrated in Figure 1. A Modelica tool (such as Dymola) will need a method for exporting the models so they can be used efficiently in the offline tool and the online system. Dymola has the option of C-code export, which is platform independent and gives models that are efficiently evaluated and easily integrated with ODE/DAE-solvers and optimizers, typically implemented in C.

NMPC software that use analytical methods for gradient computation, need the discrete-time model to provide A_k, B_k, C_k and D_k matrices (4) (typically found via sensitivity integration using the exported model, as discussed earlier). Figure 2 indicates the data flow in a discretized model component that is based on a continuous-time ODE Modelica/Dymola model, using an ODE solver which implements sensitivity integration.

Several specialized solvers for calculation of ODE sensitivities exist. For instance, CVODES (Hindmarsh and Serban, 2006) implements variable-order, variable-step multistep ODE solvers for stiff and non-stiff systems, with sensitivity analysis capabilities. For efficiency of sensitivity integration, it is a significant advantage if we have symbolic ODE Jacobians available, as we might have from a Dymola model. For models that do not provide symbolic Jacobians, we have the option of using automatic differentiation packages (CppAD, ADOLC, or others). This often requires C++ compilation.

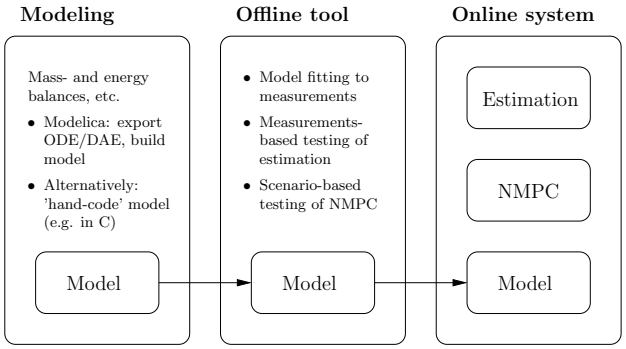

Figure 1: Overview over workflow in model usage. Data storage, data acquisition/exchange, and GUI not shown.

5 Interfacing Modelica/Dymola models with NMPC estimation and optimization software

In this section, we discuss the integration of Modelica models in NMPC software.

In the approach taken here, this involves "packaging" of the model in a "model component" that includes discretization (simulation of the model between sample intervals), such that the model is discrete time as seen from the other modules in the NMPC software package, including the state estimation and NMPC module. See Figure 2.

In NMPC software, the model component is often directly coded in a low-level language like C. While this has some advantages, for a number of reasons it is desirable to have a more user-friendly way of implementing models, using a high-level equation-based modeling language. The overall goal is to reduce the cost of modeling, which is a significant cost factor in an NMPC implementation project. Reasons for cost reduction in using a modeling language like Modelica include

- Promote reuse of models, also through building of model libraries.
- Better overview of models, ease of implementation and modifications.

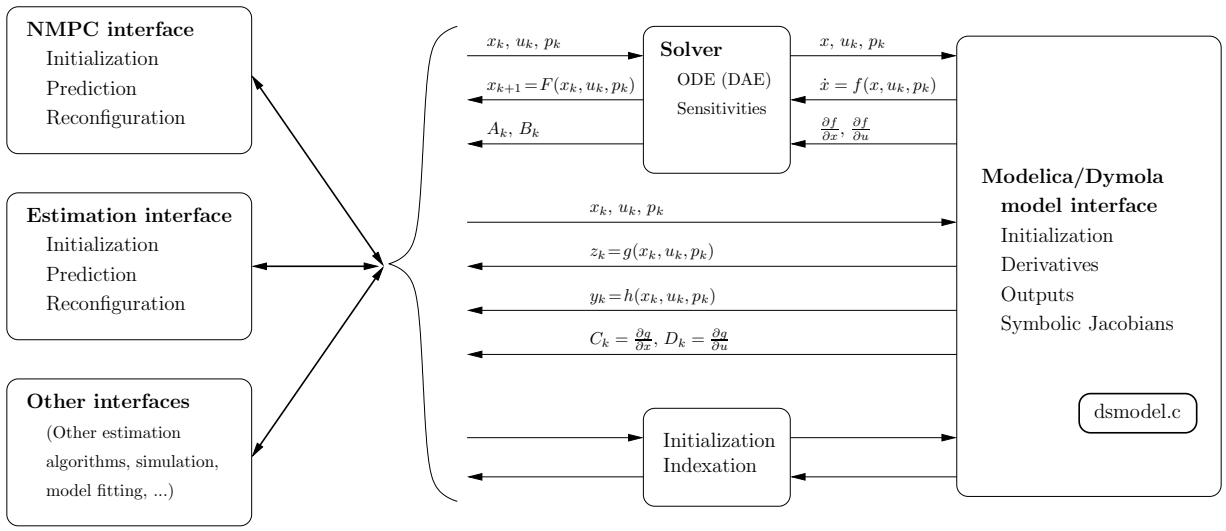

Figure 2: Overview over model component data flow.

- Easier exploitation of modeling effort in other contexts.
- Possibly easier integration of external models (external libraries, customer models, thermodynamics, etc.).

Based on this list (and other criteria as e.g. openness), Modelica is an excellent possible choice for modeling language. Moreover, the software tool Dymola provides a good Modelica modeling environment, robust and efficient algorithms for formula manipulation (tearing, index reduction, etc.) and the opportunity to integrate the models in other software, through the Dymola C-code export option.

With the C-code export, the Modelica model is available in a C-file, dsmodel.c, along with interface functions. Figure 2 illustrates how this C-file can be integrated to form a model component.

A distinct advantage of the C-code export offered by Dymola, is that it allows compilation of the total control system including model on any target system equipped with an ANSI C compiler. This is in contrast to systems which base the interface on software component interfaces such as CORBA, and requires (a version of) the modeling environment to run simultaneously.

On the other hand, it might be conceived as a disadvantage that the interface is Dymola specific, and not based on any standard. This is now being rectified with the development of the standardized FMI interface[1] for model exchange. In the newer versions of Dymola, this is now implemented, but it was not available when the work reported in this paper was performed.

[1] http://functional-mockup-interface.org/.

6 Control-relevant modeling of an offshore oil and gas processing plant in Modelica

6.1 Modeling

In the North Sea (and on other continental shelves), petroleum is produced by drilling wells into the ocean bed. From the wells, typically a stream of oil, gas and water arrives at a surface production facility (platform or ship) which main task is to separate the products. Oil and gas are exported, either through pipelines or by ship. Water is cleaned and deposited to sea or pumped back to the reservoir.

A schematic picture (in the form of a Dymola screendump) of such an offshore oil and gas processing plant is given in Figure 3. Oil, gas and water enter the plant from several sources. In reality the sources are reservoirs connected to the production facility through wells and pipelines. The separators are large tanks which split the phases oil, water and gas. The produced oil is leaving in the lower right corner of the figure, while the gas enters a compression train (not included in the figure) from the first and second separator (two leftmost tanks). Water is taken off from each separator and sent to a water treatment process.

Generally, this type of process is a fairly complex system in terms of numbers of components. However, many of the components are of the same type (mainly separators, compressors, valves, controllers, in addition to minor components such as sources, sinks, splitters, sensors, etc.), which simplifies overall modeling and make it efficient to reuse model components. Furthermore, construction of this process model benefited sig-

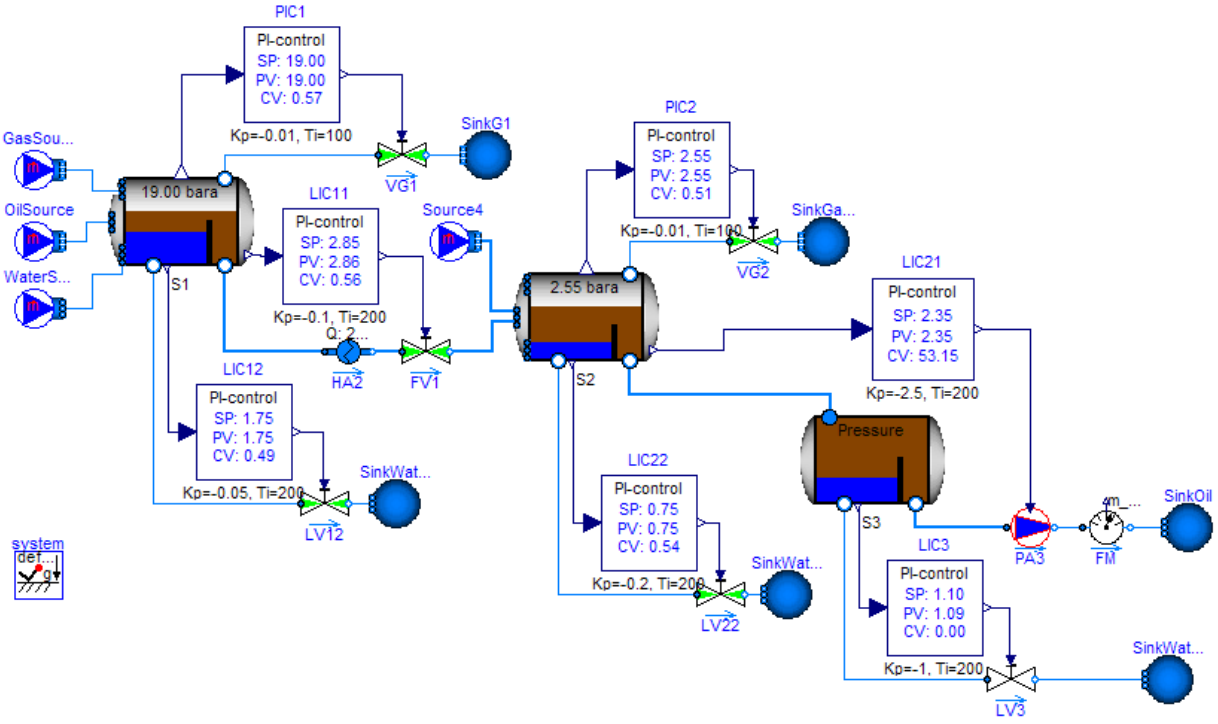

Figure 3: Overview of an offshore oil and gas processing plant, as implemented in Dymola.

nificantly from using models and concepts introduced by the new Modelica.Fluid library. Models for valves, sources, sinks, and sensors were used directly, whereas other models and functions in the new library inspired the development of our own models. For real-time efficiency reasons, we take care to ensure that we end up with an ODE-type model. The new *stream* class is an improvement to ease the construction of models satisfying this criterion.

In addition to the unit modes, medium models are necessary in order to calculate physical properties like density and heat capacity, in addition to phase transitions between oil and gas. The model should have real time capabilities, favoring simple/explicit relations. For phase equilibrium calculations, correlations of k-values (as function of temperature, pressure and molecular weight) were used together with a simplified representation of the many chemical species found in the real process. Gas density was described by a second-order virial equation, where the model coefficients were fitted to an SRK-equation for the relevant gas composition evaluated for the temperature and pressure range of current interest.

A brief description of some unit models is given below:

- **Separators**: Separators are large tanks which due to their construction, and the different densities of the components, separate water, oil and gas into different process streams. The dynamics of the separator model is based on a mass balance and flash calculations to calculate the split of oil and gas. Based on the separator geometry (and thermodynamics), water and oil levels and gas pressure can be calculated from the component masses.
- **Compressors**: The centrifugal compressor models are static models based on compressor maps (specified by the compressor vendor) of polytropic head vs. volumetric rate, parameterized in compressor speed. The compressor maps are interpolated to yield continuous relations. The compressors are strongly nonlinear, that is, the gain from compressor speed (input) to pressure and volumetric rate are strongly dependent on operating point.
- **Valves**: There are different valve models for liquid and gas flow, both based on basic valve equations. Critical and sub-critical flow are handled. The valve characteristics can be chosen to be either linear or equal percentage via a drop-down menu.

For real-time efficiency reasons, we have made an effort to develop components and modeling guidelines that ensure that the overall model we end up with is an ODE model. The main manifestation of this, is that we cannot have more than one unit that determines flow between each 'hold-up'-volume in the model. Therefore, we have in some places introduced semi-physical

'nodes', and tuned the volumes of these to retain acceptable transient response (for example, by tuning the node dynamics to be faster than the sample frequency).

6.2 Issues in preparing a Dymola model for use with an NMPC system

Before a Dymola simulation model can be used in an NMPC system, some preparation must be done. It is worthwhile to mention some of the issues involved:

- In addition to making sure that the model does not contain nonlinear systems of equations that must be solved to evaluate it (i.e, the model is an ODE as explained above), for Dymola to export symbolic Jacobians we must of course ensure that the model is differentiable. Especially if Modelica functions are used extensively, this might in some cases involve some effort.
- Dymola provides ODE-style Jacobians (A, B, C and D matrices). Unfortunately, it seems not possible to specify a subset of inputs that we want to evaluate Jacobians with respect to. This is especially critical for the B-matrix. For instance, we have in the case study in Section 8 a total of 9 possible MVs/DVs, all of which are modeled as Dymola inputs. We have chosen to control 2 or 4 of these. This means that we evaluate a B-matrix of dimension 27×9, instead of 27×2 or 27×4. This incurs considerable unnecessary complexity. If we in addition have a considerable number of parameters to be estimated also modeled as Dymola inputs, this makes the situation even worse. Future developments in the FMI interface mentioned in Section 5 might provide functionality that alleviate the situation.
- It seems the most natural way to implement communication between an NMPC system and the model, is to use 'top level' Modelica inputs and outputs. This is usually rather straightforward to implement for NMPC inputs and outputs (MVs, DVs and CVs) and measurements, but not very flexible: The Dymola C-code model interface could have been more sophisticated when it comes to identification and indexing of inputs, outputs and states. Furthermore, the use of top-level inputs and outputs can become rather awkward when it comes to model parameters that should be estimated. Again, the FMI interface might improve on this situation.

7 Case studies

7.1 State- and parameter estimation of offshore processing plant

In this case, extended Kalman filtering based on finite differences is used to estimate states and model parameters in a Modelica model of a real offshore oil and gas processing plant similar to the one illustrated in Figure 3, but with a compressor train for gas compression added. Logged data from real operation was used as measurements in this study.

The real process is fairly well instrumented, but there is no overall reconciliation of the individual measurements nor any overall measurement of key figures. From the individual measurements, most often in engineering units, it is hard to get an overview of the state of the process. With a complete process overview by the help of the model, it is possible to identify the current process state, being an essential basis for taking the correct corrective actions in case of abnormal incidents, and also essential as a starting point for optimization of process operation.

The resulting ODE model of the system was fairly stiff, with modes ranging from around 0.1 seconds to hours, while the sampling time of the process was 1 minute. Therefore, it was absolutely necessary to use an (implicit) ODE solver with varying step lengths. In this case, the CVODE ODE solver[2] was used, with Jacobians found by finite differences. For this model, with 38 states and 35 estimated parameters, the state estimation ran more than 10 times faster than real time.

The state and parameter estimation was successfully tuned and tested on data from several days of operation. An excerpt is shown in Figure 4, where the model initially is simulated 'open loop', and the state estimation is turned on after 60 minutes. The figure demonstrates, for a single compressor stage, how the compressor parameters converge such that the estimated variables match the measured ones.

7.2 Simulation study: NMPC of offshore processing plant

The case used in this section is similar to the one used in the previous section, but is based on (another) production platform. In this case, the focus is on the separation, and the gas compression is not modeled. The process has five different streams of oil and gas, that are to be separated in four separators (a separator train). In contrast to the previous case, the water phase is now

[2]From the SUNDIALS package, see http://www.llnl.gov/CASC/sundials/.

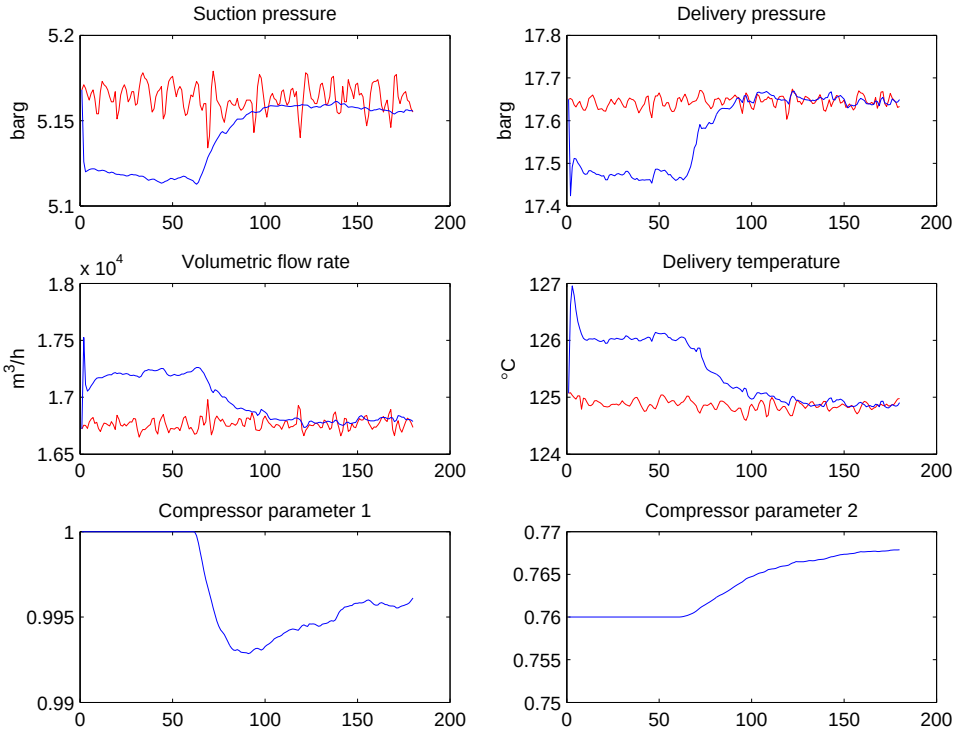

Figure 4: State and parameter estimation of one of the compressor stages. Red lines are real process data, and blue lines are estimated results. State estimation is turned on after 60 min.

explicitly modeled in the separators. The model was tuned to fit data from the real process, but all results shown in this paper are based on simulations.

The process is controlled by level controllers for water and oil, and gas pressure controllers for each separator. This is a standard solution, which works well in many/normal cases. However, in some cases, disturbances in the inlet flows from the inlet pipelines/wells can cause problems for the control of the separators. The levels in the separators will vary, which may cause bad separation and may be detrimental for equipment downstream of the separators, due to uneven flow out of the separator train. The purpose for this study is to see if NMPC with state and disturbance estimation, using the level controller setpoints as manipulated variables (MVs), can exploit the buffer capacity in the separators to smooth out the outlet flows of water and oil. The oil is in this particular case entering a distillation column, and the water is entering a glycol regenerator, for regeneration of glycol that is added in the process. Smoother inflow to these units may allow more regular/increased production of the overall process.

There are six manipulated variables: The setpoints for water and oil level controllers in the separators (two of the separators does not separate water, and hence does not have a water level controller). The controlled variables (CVs) are pressures, levels and valve openings for all separators, and rate of change of glycol concentration in one separator.

The resulting model, with 29 states, was not particularly stiff. Therefore, a simple forward Euler ODE solver was used in this case. The NMPC system, including state and disturbance estimation based on finite differences, and NMPC optimization with gradients found by finite differences, ran considerably faster than real time, using a sample interval of 6 s.

Some simulation results with a disturbance, a time-limited increased flow in one of the inflowing pipelines, are shown in Figures 5–7. Figure 5 shows how the NMPC reduces the level controller setpoints in the inlet separator (resulting in increased outflow valve openings, see Figure 7), to let the increased inlet flow (detected by the state and disturbance estimation) be smoothed out over all the separators. Figure 6 demonstrates how the NMPC achieves smoother outflow from the last separator, and that the glycol fraction in the water varies less.

8 Evaluation of gradient computations

This section aims to illustrate the advantages and differences between finite differences and sensitivity-based

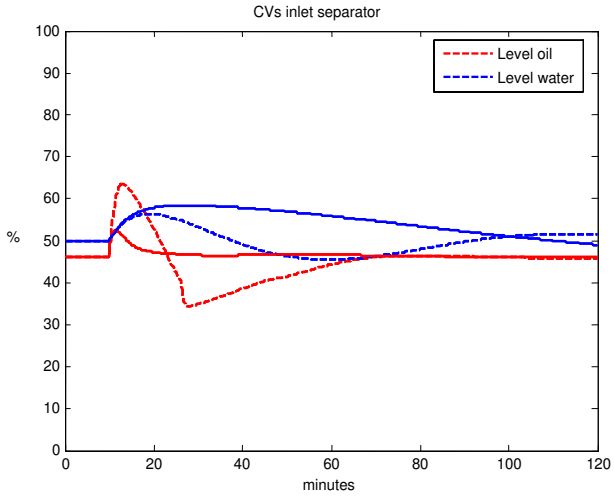

Figure 5: Oil and water levels in the inlet separator, with MPC (solid) and without MPC (dashed).

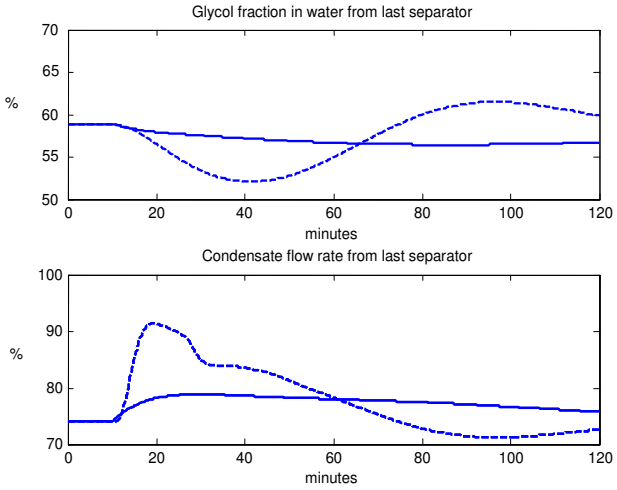

Figure 6: Glycol concentration in glycol/water mixture (top) and oil flow rate (bottom) from the last stage separator, with MPC (solid) and without MPC (dashed).

sensitivity computations. The model we use as NMPC prediction model (cf. Figure 3) has 27 states. We consider two NMPC problem formulations, one being 2×2 (2 MVs and 2 CVs), the other 4×4.

Strictly speaking, the results in this section only apply to this specific case, but we believe there is considerable generality in the trends reported. Issues that will be discussed, are computational complexity (timing), accuracy, and implementational aspects. The results are of course influenced by many factors not investigated (i.e., kept constant) here, as for example number of states, stiffness, exact definition of input blocks, etc.

We use a Matlab interface to the NMPC system, and choose to compare computational complexity by measuring execution time in Matlab. Although this has some drawbacks, it should give a fairly accurate picture of the relative performance. To increase the reliability, for each recording of execution time we run 5 consecutive identical NMPC scenarios, and record the smallest execution time (based on wall clock time). This execution time includes the Kalman filter and NMPC optimization, but as the gradient computation is the most computationally expensive part, and the other parts are independent of choice of method for gradient computation, the difference in execution times should give a fairly good estimate of the difference in complexity of gradient computation.

8.1 Correctness and accuracy

It is clear that using finite difference (from now on FD) and analytic sensitivity methods based on sensitivity integration (AS) should give the same gradients "in the limit" (of perturbation-size and integration tolerances). Nevertheless, it is of interest to test this, also to get a feel for how large errors (or differences) relaxed integration tolerances and realistic perturbations will lead to.

From Figure 8, we see that for small integration error tolerances, the gradients found are fairly correct in both methods, but the error in the FD sensitivity matrix increases much faster as the error tolerances are increased. An important note regarding the implementation of the AS-method is that we have chosen to have error control on both states *and* sensitivities, not merely states. In our experience, this can be essential to ensure accurately enough sensitivities when using AS.

We do not discuss here the choice of perturbation size in FD methods, as this does not differ from the general discussion in e.g. Nocedal and Wright (2006). Suffice it to say that the general trends in Figure 8 are fairly independent of choice of perturbation size.

As a side-remark, even though both methods only give correct gradients in the limit, FD methods gives a direct approximation to the gradient of the objective function that is actually being optimized (including integration errors). This can in theory be an advantage in the line-search step of SQP algorithms.

8.2 Computational complexity

In this section we will compare the computational cost of different gradient computations as the number of NMPC degrees of freedom increases. We increase the degrees of freedom both in number of input blocks as

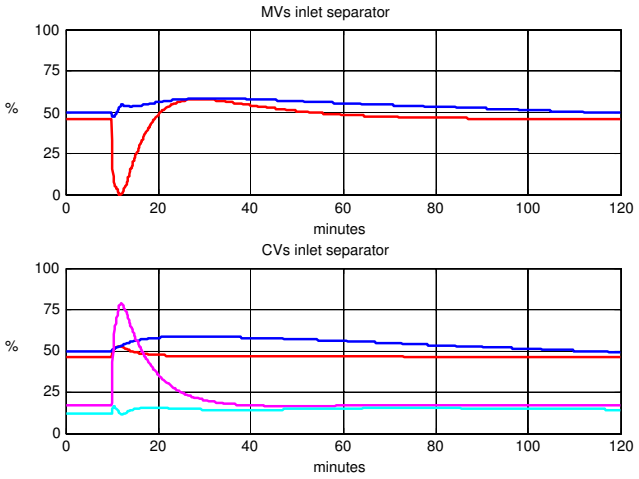

Figure 7: Level controller setpoints (MVs, top) and level and valve openings (CVs, bottom) in inlet separator. Red line is oil, blue line is water, magenta is oil valve opening, cyan is water valve opening.

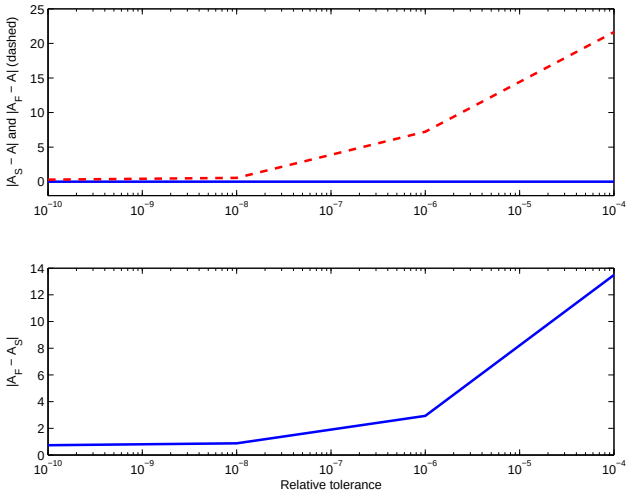

Figure 8: Top: Difference between 'real' (as found by AS methods with tolerances of 1e-12) step response matrix A and step response matrices for higher tolerances. Bottom: Step response matrices using AS and AF (almost) converge as tolerances decreases.

well as number of inputs. The cases we will compare, are gradient computation using finite differences (FD) with or without using symbolic Jacobians in the ODE solver, and analytical gradient computation using sensitivity integration (AS), also with and without using symbolic Jacobians.

We use the same ODE tolerances in all cases, and for sensitivity integration the sensitivities are included in the error control. The relative time usage for different number of inputs and input blocks are shown in Figure 9. In Figure 10, the experiment is repeated for $N_u = 2$, but without using symbolic Jacobians in the ODE solver.

The following observations are made:

- FD grows approximately quadratically in input blocks (as additional input blocks incurs both additional perturbations and ODE solver resetting), while AS grows approximately linearly in input blocks (incurs only additional ODE solver resetting). Note the logarithmic scale in Figure 9. To the extent that this is general, this means that AS will always outperform FD when many input blocks are used.
- Increasing number of input blocks (leads to more frequent ODE solver resetting) is more expensive than increasing number of inputs (leads to larger "sensitivity state") when using AS. We attribute this both to the efficiency of CVODES in exploiting the structure in the sensitivity equations, but also to the next issue:
- Increasing number of inputs does not significantly increase complexity of AS. This may be surprising, but can be explained in this case by the fact that all ODE Jacobians are calculated irrespectively of how many of the inputs are actually active, as discussed in Section 6.2. If Dymola allowed calculation of only those Jacobians that are needed, this could considerably speed up execution time.
- AS suffers significantly more than FD from not having symbolical Jacobians available (Figure 10).

Finally, we mention that in our experience, implementing Modelica-functions in C can significantly speed up FD (not done in the case in this Section), see also Section 9.3. If this can be combined with export of symbolic Jacobians, it can also speed up AS, but to a much less extent. In other words, implementing Modelica-functions in C is less important when using AS.

9 Experiences with using Modelica and Dymola for real time process control applications

In this section, we summarize some of our experiences with using Modelica and Dymola for process control applications.

9.1 Modelica modeling in Dymola

When it comes to modeling, Modelica and Dymola has much to offer over implementing the models in C. Due

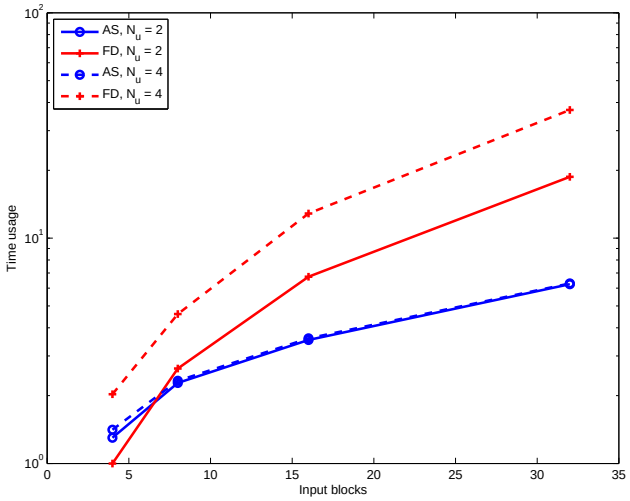

Figure 9: Relative time usage for different number of inputs and input blocks.

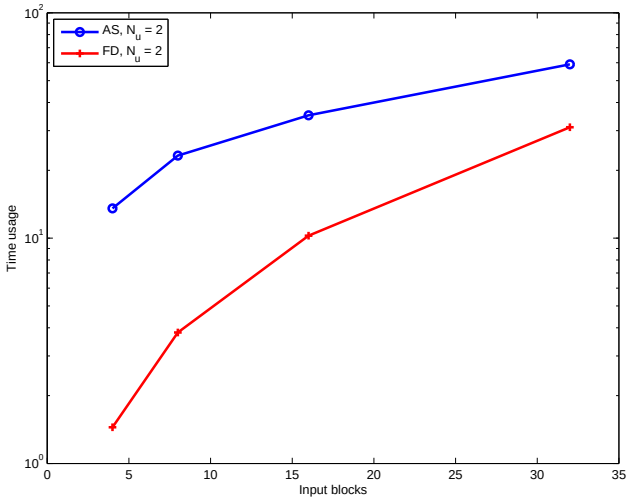

Figure 10: Relative time usage for different number of inputs and input blocks, without exporting symbolic Jacobians from Dymola.

to the object orientation and the graphical interface it is easy to work on details and at the same time have an overview over the whole model. Using a tool such as Dymola, tasks like manipulation, testing and simulation of the model are convenient.

In this paper, we have used cases from offshore oil and gas production. We saw advantages in terms of reuse between these projects, and we expect the rewards to be even greater at later stages. Using an object-oriented environment like Modelica makes it easier to develop unit models with more general interfaces, such that they are easier reused. The Modelica Standard Library, in particular the Modelica.Fluid library, provides a good basis for doing this in many process control applications, including the ones reported here. Some of our unit models were inspired by Modelica.Fluid components, and by drawing inspiration from Modelica.Media, we had a convenient structure for implementing the thermodynamics.

As with other equation-based modeling systems, debugging models during model development is a challenge in Dymola, and tools to help model debugging would be a benefit. However, by testing unit models thoroughly before aggregating them, many problems can be avoided.

When using models with nonlinear equation systems (DAEs), we had in some cases problems with initialization of the nonlinear equation systems, and identifying which variables that were part of the equation system (cf. discussion in Section 2.2). Of course, when making sure the model was an ODE, these problems were avoided.

9.2 Integration of Modelica/Dymola models in NMPC software

Using the C-code export option of Dymola, it is fairly straightforward to integrate the Modelica models as model components as described in Section 5.

A significant part of the effort in constructing the model component based on the structure illustrated in Figure 2, is to generate and keep up to date the referencing/indexing variables. This information is necessary in the NMPC user interface, for instance for tuning of the EKF and the NMPC controller. This is considerable and errorprone work if done by hand, but recent developments (the new FMI interface) promise some improvements in the model interface to automate this.

In some cases, it would be an advantage to be able to debug the model code. Due to the structure of the auto-generated code, this is hard.

9.3 Running Modelica/Dymola models in NMPC software

There are some further interesting findings from the case study in Section 7.2. We had this model implemented as a model component in C before we implemented it in Modelica. By using profiling tools, we found that running NMPC with the model component based on the Modelica model, used less than 20% additional time compared to using the pure C model component, where most of the difference must be attributed to Modelica overhead since the models were practically mathematically identical.

However, to get the Modelica-based model to run this fast, we had to implement the Modelica functions used in the Modelica-model in C. Not surprisingly, there is considerable overhead in the implementation of Modelica functions, especially related to indexing of arrays. The possibility to implement Modelica functions in C is supported by the Modelica specification, and implemented in Dymola, and can be a considerable practical advantage for real time applications.

10 Concluding remarks

In our experience, the use of Modelica/Dymola for modeling for NMPC purposes shows significant promise. Such environments are helpful in developing complex process models, towards reuse of unit models, and we see potential for increased model value (by extending the application area of the model) and easier customer participation in model development.

However, using Modelica/Dymola models for NMPC has some hurdles. Some effort is required to make a Modelica simulation model ready to be used with NMPC software, but further development in the software interfaces may reduce these difficulties.

We found that constructing the sensitivity matrix using analytical methods becomes significantly faster than finite difference-based methods as the number of inputs and/or input blocks increases, and therefore such methods are important as models get larger. Moreover, we conclude that to use analytical methods, we should have ODE Jacobians available, either symbolically or automatically.

Finally, we emphasize that process models for NMPC should be developed with the specific task in mind, in terms of issues such as complexity, accuracy and smoothness. In some cases, this means that the model should be an ODE, while models from component-based modeling languages such as Modelica naturally translates into DAEs. It will in general require some effort and compromises for Modelica models to translate into ODEs.

Acknowledgments

The authors thank Statoil AS for providing information and input to the case studies.

References

Biegler, L. Efficient solution of dynamic optimization and NMPC problems. In F. Allgöwer and A. Zheng, editors, *Nonlinear Predictive Control*, pages 219–245. Birkhauser, Basel, 2000.

Biegler, L. T., Cervantes, A. M., and Wächter, A. Advances in simultaneous strategies for dynamic process optimization. *Chem. Eng. Sci.*, 2002. 57:575–593. doi:10.1016/S0009-2509(01)00376-1.

Bock, H. G., Diehl, M., Leineweber, D. B., and Schlöder, J. P. A direct multiple shooting method for real-time optimization of nonlinear DAE processes. In F. Allgöwer and A. Zheng, editors, *Nonlinear Predictive Control*, volume 26 of *Progress in Systems Theory*, pages 246–267. Birkhäuser, Basel, 2000.

Findeisen, R., Imsland, L., Allgöwer, F., and Foss, B. A. State and output feedback nonlinear model predictive control: An overview. *European J. of Control*, 2003. 9(2-3):190–206. doi:10.3166/ejc.9.190-206.

Foss, B. A. and Schei, T. S. Putting nonlinear model predictive control into use. In *Assessment and Future Directions Nonlinear Model Predictive Control*, LNCIS 358, pages 407–417. Springer Verlag, 2007.

Hairer, E., Nørsett, S. P., and Wanner, G. *Solving Ordinary Differential Equations I – Nonstiff problems*. Springer-Verlag, 2nd edition, 1993.

Hindmarsh, A. C. and Serban, R. *User Documentation for CVODES v2.5.0*. Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, 2006.

Imsland, L., Kittilsen, P., and Schei, T. S. Model-based optimizing control and estimation using modelica models. In *Proc. of Modelica'2008*. Bielefeld, Germany, 2008.

Imsland, L., Kittilsen, P., and Schei, T. S. Using modelica models in real time dynamic optimization – gradient computation. In *Proc. of Modelica'2009*. Como, Italy, 2009.

Jørgensen, J. B. Adjoint sensitivity results for predictive control, state- and parameter-estimation with nonlinear models. In *Proceedings of the European Control Conference, Kos, Greece*. 2007.

Lang, Y. D. and Biegler, L. T. A software environment for simultaneous dynamic optimization. *Computers & Chemical Engineering*, 2007. 31(8):931–942. doi:10.1016/j.compchemeng.2006.10.017.

Li, W. C. and Biegler, L. T. Multistep, Newton-type control strategies for constrained, nonlinear processes. *Chem. Eng. Res. Des.*, 1989. 67:562–577.

Maciejowski, J. M. *Predictive Control with Constraints*. Prentice-Hall, 2001.

Nocedal, J. and Wright, S. J. *Numerical Optimization*. Springer-Verlag, New York, 2006.

Nørgaard, M., Poulsen, N. K., and Ravn, O. New developments in state estimation for nonlinear systems. *Automatica*, 2000. 36(11):1627–1638. doi:10.1016/S0005-1098(00)00089-3.

Qin, S. J. and Badgwell, T. A. A survey of industrial model predictive control technology. *Control Engineering Practice*, 2003. 11:733–764. doi:10.1016/S0967-0661(02)00186-7.

Rawlings, J. B. and Mayne, D. Q. *Model Predictive Control: Theory and Design*. Nob Hill Publishing, Madison, WI, 2009. 576 pages, ISBN 978-0-9759377-0-9.

Ringset, R., Imsland, L., and Foss, B. A. On gradient computation in single-shooting nonlinear model predictive control. In *Proc. of IFAC DYCOPS 2010, Leuven, Belgium*. 2010.

Schei, T. S. A finite-difference method for linearization in nonlinear estimation algorithms. *Automatica*, 1997. 33(11):2053–2058. doi:10.1016/S0005-1098(97)00127-1.

Schei, T. S. On-line estimation for process control and optimization applications. In *Proc. 8th International IFAC Symposium on Dynamics and Control of Process Systems (DYCOPS-07)*. Cancún, Mexico, 2007.

Schei, T. S. On-line estimation for process control and optimization applications. *Journal of Process Control*, 2008. 18:821–828. doi:10.1016/j.jprocont.2008.06.014.

Schlegel, M., Marquardt, W., Ehrig, R., and Nowak, U. Sensitivity analysis of linearly-implicit differential-algebraic systems by one-step extrapolation. *Applied Numerical Mathematics*, 2004. 48:83–102.